IOT-BASED SMART PARKING SYSTEM

PHASE 4

DEVELOPMENT PART 2

BLOCK DIAGRAM:

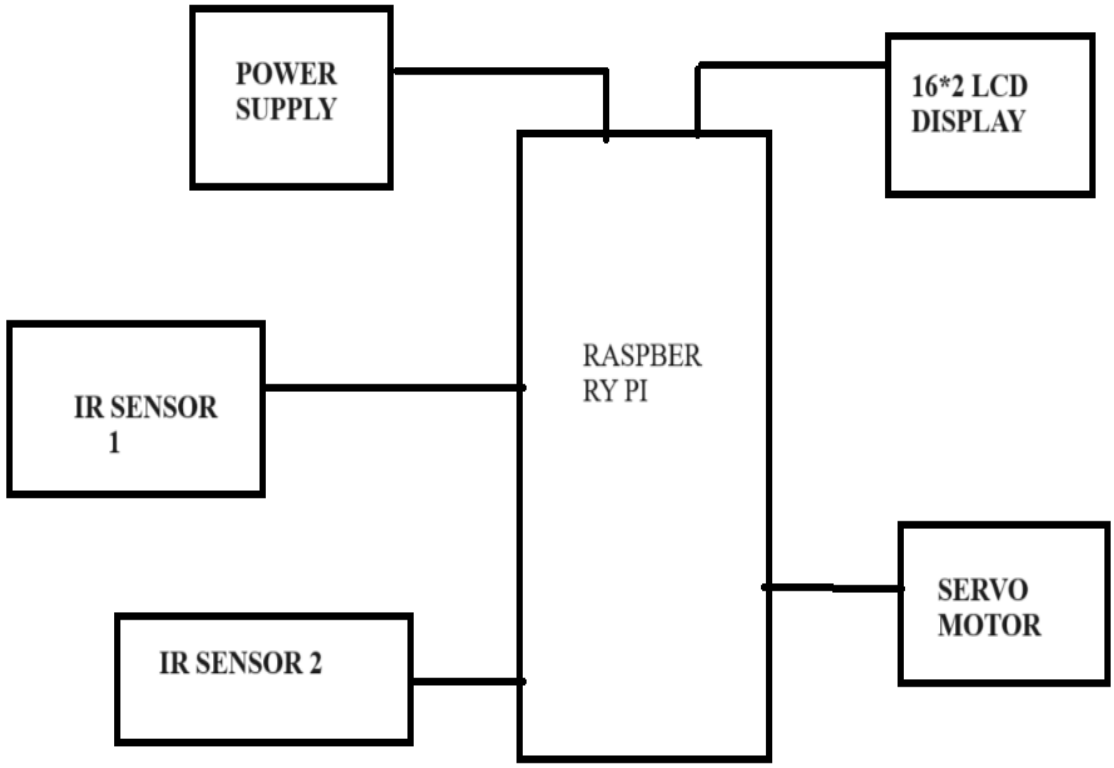

Figure 1:Block diagram

EXPLANATION:

- The IR Sensor continuously monitors the area and sends signals to the Raspberry Pi.
- The Raspberry Pi processes these signals and makes decisions based on the presence or absence of objects detected by the IR Sensor.
- If access is granted, the Raspberry Pi sends a signal to the Servo Motor to open the barrier.
- The WiFi module allows the Raspberry Pi to connect to a network, which in turn enables communication with external devices or servers. For example, an app or a server could be set up to control the system remotely.
- The LCD Display provides a visual interface to convey messages or system status to users.
- The barrier (gate or door) is controlled by the Servo Motor. When the Raspberry Pi sends a signal to open or close, the motor physically moves the barrier accordingly.
- An optional app or server component can be added for remote control and monitoring. This could allow for features like opening the barrier from a mobile device or receiving notifications.

MOBILE APPLICATION DEVELOPMENT:

Designing a high-performing and efficient system will be difficult if we don't follow a process. We have created a step-by-step process to create a web IoT-based mobile parking system. This will include the design creation and feature selection too.

Figure 2

1. STUDY THE MARKET, COMPETITION AND TARGET USERS

- The market for parking management systems is quite specialized. Vehicle users exist. On the basis of corporate, recreational, and commercial car parking, we may further specialize.
- Once we are aware of the goals of wer project, we must research wer audience. What kind of communication methods are they employing?
- Determine the parking difficulties they are presently experiencing. Know who the direct rivals are in this market. We'll discover that using this knowledge will enable we to improve wer concept and produce the ideal resolution.

2. DEFINE THE CORE PURPOSE OF THE PRODUCT

- We may sketch wer app concept and create wer vision with the aid of the core. We'll be able to reinvent the solution with the aid of the application's primary goal.
- Knowing why we are building the application is the first step. Identify the current systems, the problems the users are having, and the ways in which wer proposal may benefit them.
- This will aid in enhancing the application's vision and lowering the aims or goals.

3. VALIDATE THE CONCEPT AND DEFINE THE BLUEPRINT

- It is simpler to convince the audience of the validity of wer mobile parking system proposal if it is written down and is obvious. Find out if wer workforce is prepared to utilize wer proposed application.
- Knowing the application's conversion rate and downloads need validation. Choose a strategy for developing the application after it has been verified.
- We should include information in wer blueprint on the app's conception and design planning. We could wish to specify the technology stack, development process, and strategy.

4. DETERMINE THE UNIQUE FEATURES

- This is a crucial stage in the creation of wer application. The characteristics that are phase-wise should be defined. For the development of prototypes, for instance, certain qualities are crucial. Keep all the complicated features until the last phase of development of wer sophisticated software.

5. WORK WITH UI/UX TEAM AND CREATE A WORKABLE PROTOTYPE

- Work with the UI/UX team when the plan and features are prepared. Wer design should be compatible with wer target audience. Understand their requirements, app usage habits, and expected behavior when using wer application.

- Gaining interest and creating an immersive application depend on the UX approach. Plan for user behavior such as one-handed holding, landscape displays, and other actions.

6. DEVELOP THE DESIGNS USING CLEAN CODES

- It is now time to turn wer designs into a workable solution based on the UI and UX strategy that we have already completed. Smooth communication between the client and server ends should be guaranteed by the backend staff.
- Wer code must be well-organized and documented. This can also be useful for testing and maintenance. Clean coding might assist the developer even when we are planning an upgrade.

7. TEST THE APPLICATION BEFORE LAUNCHING

- We can manage the application launch on schedule with the use of a test-driven strategy. We might wish to look for reusability, coding errors, and other problems with the program.
- We might also wish to test the item and its features to make sure they operate as intended. Determine every test we must run before releasing the program.
- Create test cases and a plan for each test. Make care to test the components or functions until they perform as intended.

8. PLAN FOR AFTER-SALES SUPPORT

- We may improve wer application with the aid of customer feedback. For the mobile application, we should have a strategy for optimization, an upgrade based on user input, and support.
- Be careful to include routine sensor and application maintenance in wer duty chart.

FLUTTER-BASED PARKING SYSTEM APP:

We'll make use of Flutter, a well-liked mobile app development framework, to create the mobile app for the smart parking project. With Flutter, we can produce natively produced desktop, web, and mobile applications from a single codebase.

The steps for making the app are as follows:

Create Flutter:

- Follow the directions on the official Flutter website to install Flutter.
- Run “flutter doctor” in the terminal to check the installation.

New Flutter Project Creation:

- Run flutter in wer terminal. “smart_parking_app”be made.

Project Organization:

- Open the ‘lib’ directory. We will enter wer Dart code here.

Create the UI:

- ‘Lib/main.dart’ should be opened. This is where wer app starts.
- Create the user interface to show information about parking availability. Using Flutter widgets, we may ‘Container’, ‘Column’, ‘ListView’, etc. to create wer UI.

SOURCE CODE:

```
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
```

```
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: ParkingAvailabilityScreen(),
    );
  }
}

class ParkingAvailabilityScreen extends StatefulWidget {
  @override
  _ParkingAvailabilityScreenState createState() =>
      _ParkingAvailabilityScreenState();
}

class _ParkingAvailabilityScreenState extends State<ParkingAvailabilityScreen> {
  // Add logic to receive and display parking availability data here

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Parking Availability'),
      ),
      body: Center(
        child: // Add widgets to display parking availability data here
      ),
    );
  }
}
```

Linking to the Raspberry Pi

In order to connect between the Raspberry Pi and the mobile app, use a protocol like MQTT, WebSockets, or HTTP. In the '_ParkingAvailabilityScreenState', implement the appropriate logic.

SOURCE CODE:

```
import 'package:http/http.dart' as http;


class _ParkingAvailabilityScreenState extends State<ParkingAvailabilityScreen> {
  String availabilityData = 'Loading...';


  void fetchParkingAvailability() async {
    var response = await http.get('http://raspberry-pi-ip:port/parking_data');
    setState(() {
      availabilityData = response.body;
    });
  }


  @override
  void initState() {
    super.initState();
    fetchParkingAvailability();
  }


  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Parking Availability'),
```

```
      ),
      body: Center(
        child: Text(availabilityData),
      ),
    );
  }
}
```

Test the App:

Use the command flutter run to launch the application on a computer or actual hardware.

Replace Dummy Data:

The endpoint where wer Raspberry Pi is delivering parking availability data should be substituted for the URL 'http://raspberry-pi-ip:port/parking_data'.

Real-time updates are handled:

- we might need to develop a system to get real-time updates from the Raspberry Pi, depending on the connection protocol we're using.

- Keep in mind that this is only a simple template to get we going. Depending on wer unique needs, we'll need to modify and build upon it in order to add user authentication, manage edge situations, improve UI/UX, and perform real-time updates.

- Additionally, make sure wer Raspberry Pi is configured to deliver information about parking availability through the designated endpoint

EXPERIMENTAL RESULTS

A.Sign-in Page:

This is the Sign-in page where we can login with Mobile Number and get OTP verification. Here admin and user Sign-in through same page as shown in Figure 3.

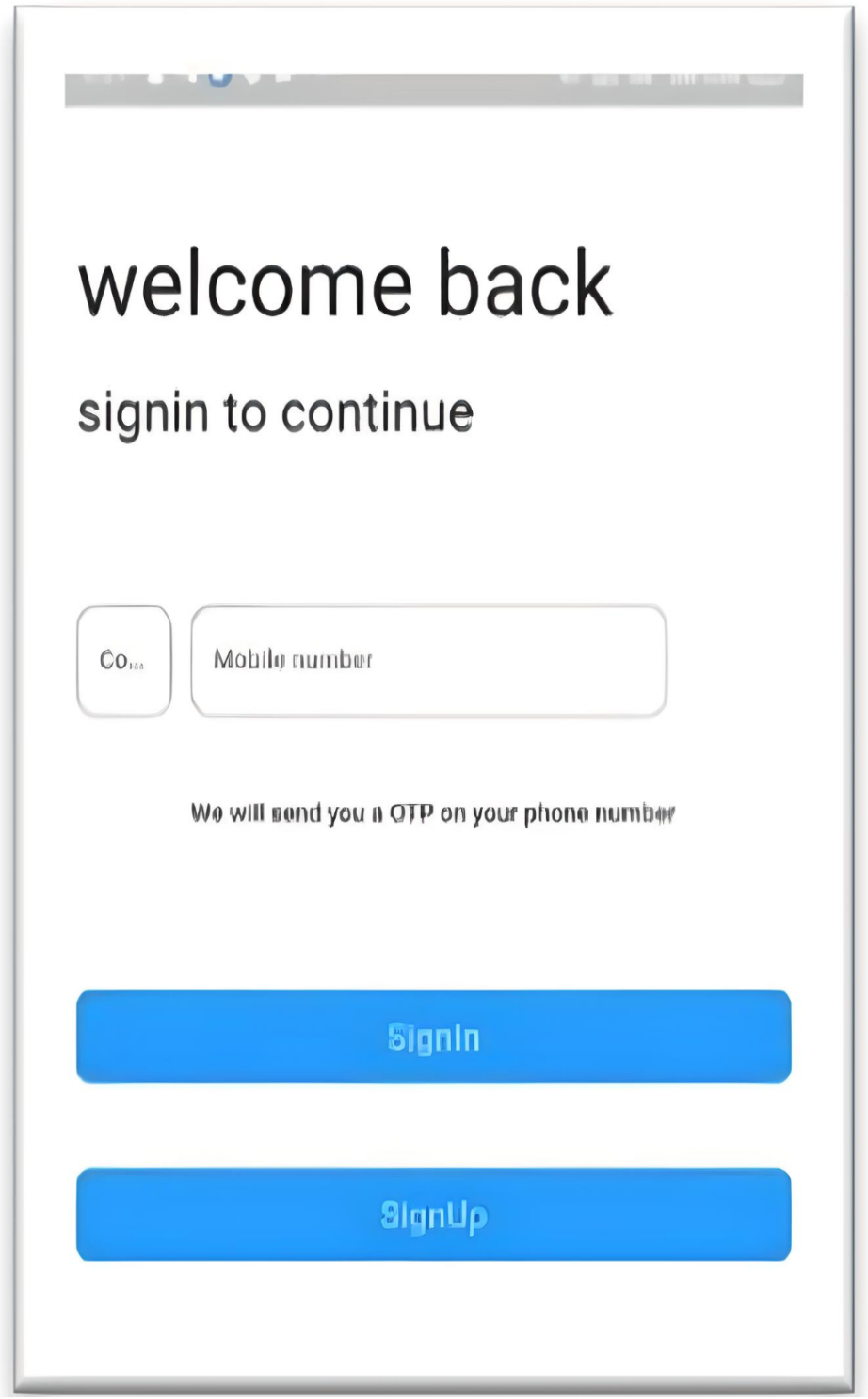

Figure 3: SIGN IN PAGE

B.Finding Parking Slot

In this Find Parking Slot as shown in Figure 4 the users can see their nearby parking slot and their location using Google API

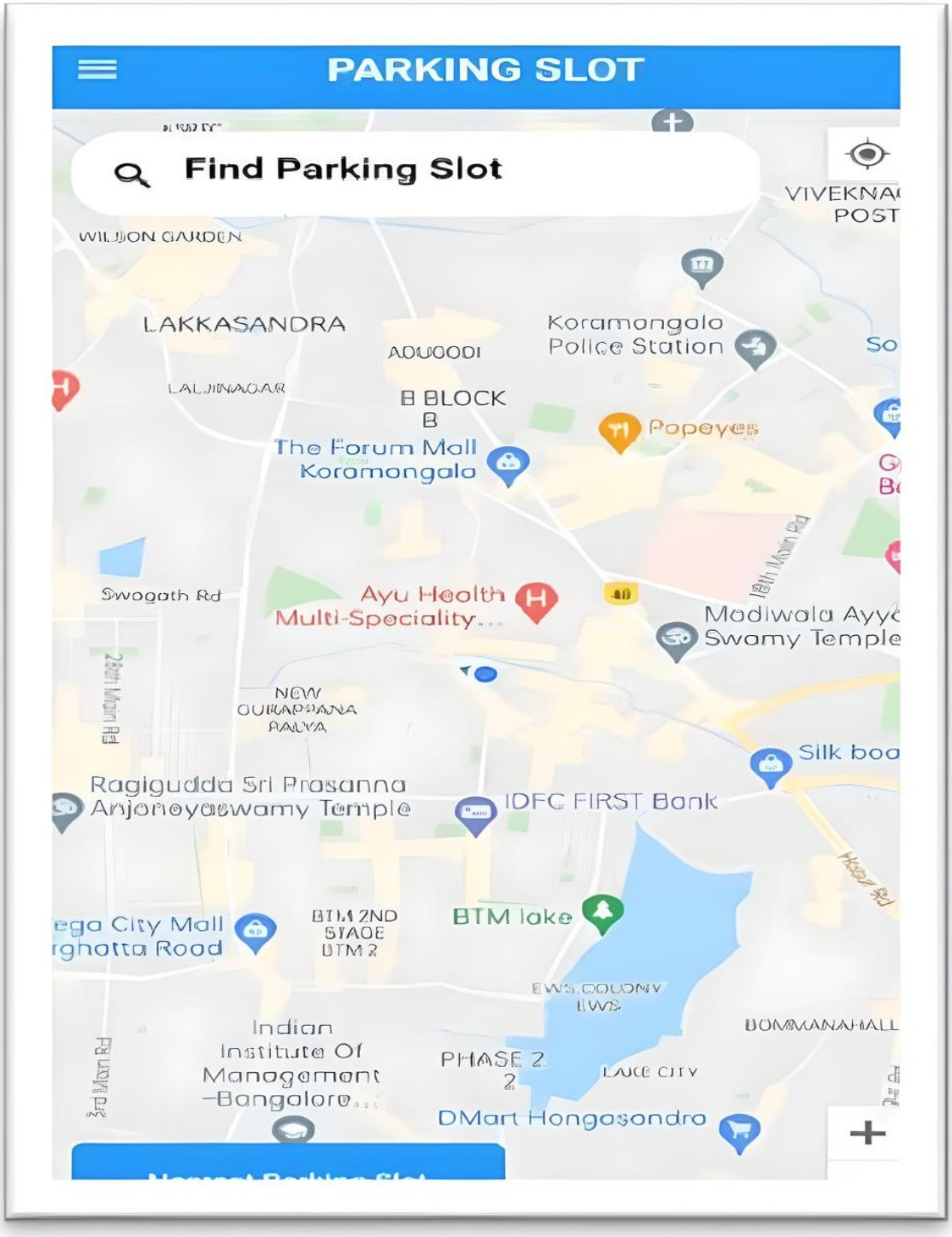

Figure 4: Find Parking Slot

C.Profile Page

As shown in Figure 5, this is profile page where the details of the users and the booking details can be viewed and edited**.**

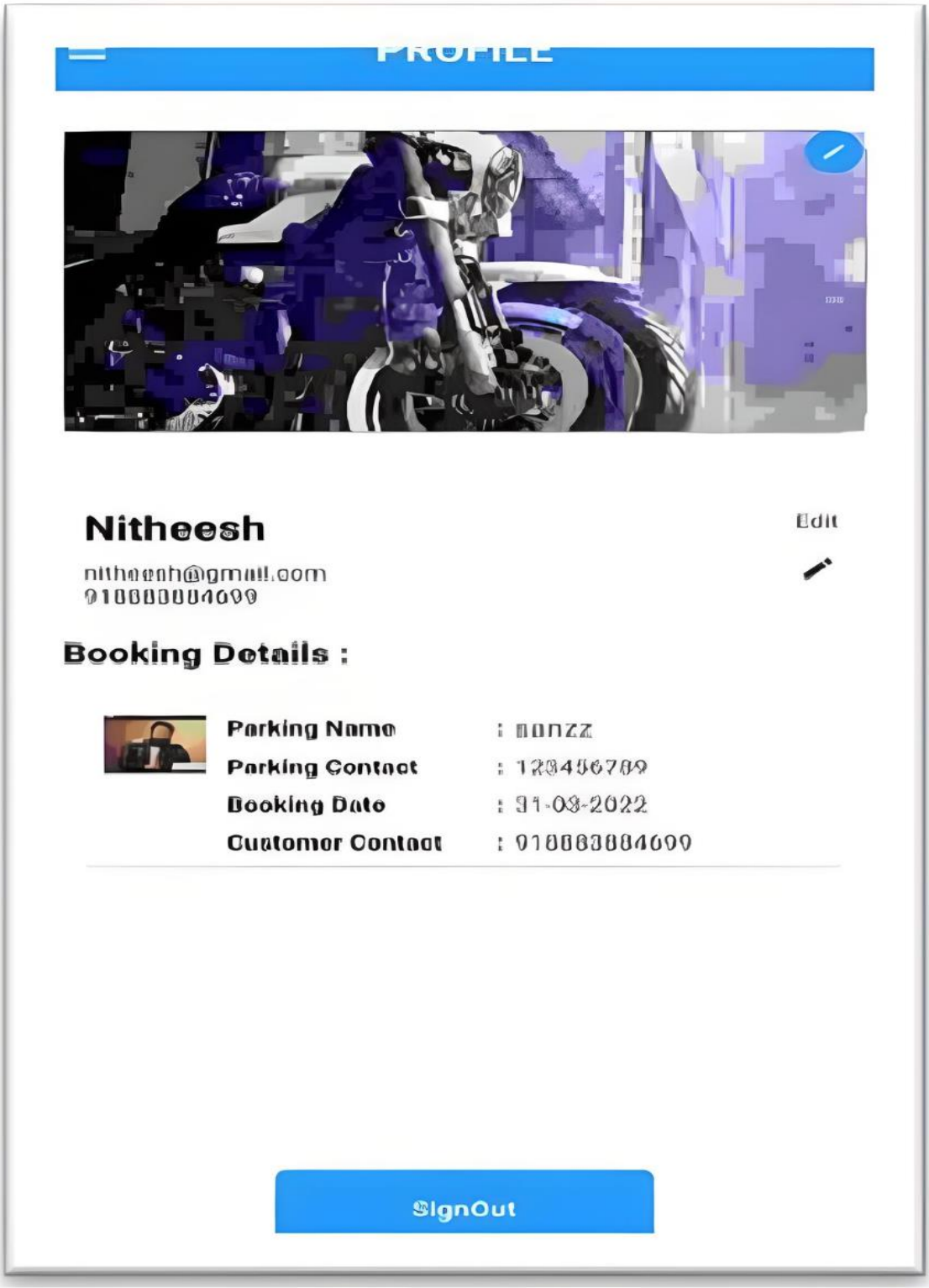

Figure 5: profile page

D.Admin-Verification Page

As shown in Figure 6, this page is used for Admin to check the parking slot status of the users so that admin can verify**.**

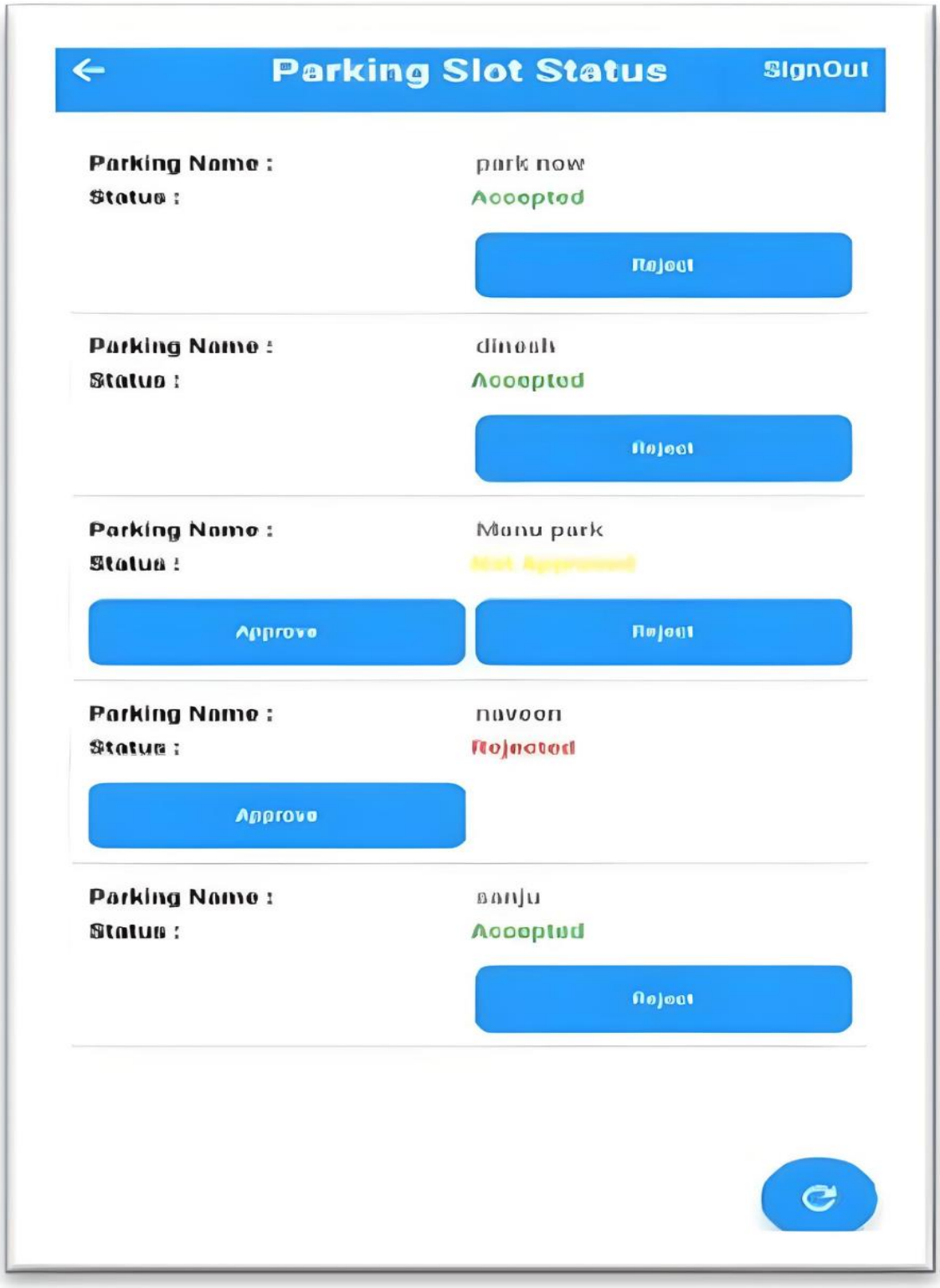

Figure 6: Admin-Verification Page

THE PROBLEM SMART PARKING APP SOLVES:

The objectives of this app are as follows :

• To make parking lots more efficient.

• Reduce chances of theft significantly.

• Reduce the time taken for check-in and check-out of vehicles.

• Display the necessary info and the current status of the parking lot.

• Save paper and go digital.

The functions of this app are as follows :

• A smart parking system wherein the driver can check-in and check-out via displaying a unique QR code.

• The system is implemented within an application made using Flutter.

• User needs to either Sign-In or Sign-Up for using the app and his information is to be stored and verified using the Cloud FireBase database and FirebaseAuth Authentication.

• Once the user has logged in, he can generate his unique QR code for displaying at the parking lot and either check-in or check-out.

• Unique QR codes to be generated for each user and each parking lot and their information should be stored in a database using Cloud Firestore.

• Once the user gets his QR code scanned and checks in, his data(including his vehicle's information) is stored in a dynamic database using Cloud Firestore and the details of the parking lot and available parking spots in the parking lot are displayed in the app.

• Now, only the user who has checked-in can check out that particular vehicle. During check-out, compare the user data with the already stored data and check-out only if match is found and then delete that entry.

The main goal of this app is to contribute in building a more advanced world and ofcourse Digital India.

Challenges we ran into

• The main hurdle while developing this app has been handling the authentication and database, namely Firebase and Cloud Firestore.

• The app works on an emulator as of now, which was quite tricky to handle. We had trouble in the beginning understanding how everything links up, but we got it while working on the project further.

• Making the security of vehicles better was quite a thoughtful but we finally came up with an idea and had it working.